Buză Elena-Raluca

Grupa 331AA

Tema 1 – Tehnici de invatare automata

Recognizing handwritten digits

1. Descrierea proiectului

Aceasta aplicatie isi propune sa ilustreze procesul de recunoastere a cifrelor scrise de mana prin aplicarea unui algoritm de invatare automata, anume K-Nearest Neighbors (KNN). Recunoasterea cifrelor scrise de mana reprezinta o problema fundamentala in invatarea automata, avand aplicatii in recunoasterea optica a caracterelor si procesarea documentelor digitale.

Obiectivul principal este de a explica abordarea utilizarii algoritmului KNN pentru a clasifica cu precizie cifrele scrise de mana din setul de date.

2. Setul de date

Setul de date contine 10 foldere, pentru fiecare cifra (0 .. 9). Fiecare folder contine un numar variabil de imagini cu extensia .png, fiecare imagine fiind etichetata. Setul de date a fost obtinut prin descarcarea imaginilor corespunzatoare fiecarei cifre. Exista un numar mare de imagini in setul de date, tocmai pentru ca antrenarea algoritmului sa se faca intr-un mod corect.

Datele sunt relevante temei alese: avem imagini cu toate cifrele. Aceastea sunt etichetate, fiecare imagine avand un nume/eticheta unic, neexistand duplicate. Sunt organizare din punct de vedere structural, caci avem 10 foldere care contin doar cifre aferente index-ului folderului.

Setul de date este divizat in 3 subseturi:

- Antrenare: 75% din total
- Validare: 1000 pentru fiecare folder (din cele de antrenare)
- Testare: 25% din total

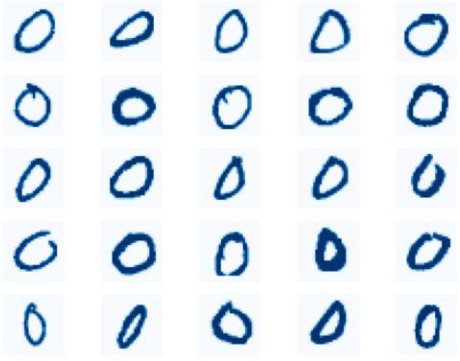
Pentru fiecare folder, reprezentand o cifra intre 0 si 9, avem urmatoarele dimensiuni pentru subseturile de date (numar de imagini):

Cifra	Antrenare	Validare	Testare
0	4923	1000	980
1	5742	1000	1135
2	4957	1000	1032
3	5131	1000	1010
4	4842	1000	982
5	4421	1000	892
6	4918	1000	958
7	5263	1000	1028
8	4851	1000	974
9	4949	1000	1009

Se poate observa cum pentru fiecare folder s-a incercat mentinerea unui numar aproximativ egal de imagini cu cel al celorlalte foldere, astfel ca antrenarea si testarea sa se realizeze in mod corect, pentru a nu exista diferente majore intre foldere.

Setul de date contine 69997 de esantioane, impartite in 10 foldere si 3 subseturi, conform tabelului de mai sus. Acest set de date are o dimensiune mare tocmai pentru a se realiza o antrenare corecta a algoritmului KNN, in vederea obtinerii unei acuratete cat mai mari.

Exemple de imagini din setul de date:



3. Preprocesarea si normalizarea datelor

Fiecare imagine a fost transformata intr-un vector de octeti cu valori cuprinse intre 0 si 255. Astfel, pentru fiecare subfolder, train, validation si test, am realizat 2 vectori, X si Y, care contin octetii imaginii, respectiv eticheta fiecarei imagini.

Avem urmatorii vectori: X_train, Y_train; X_validation, Y_validation; X_test, Y_test.

Deoarece vectorii Y contin doar etichetele imaginii, am scalat doar vectorii X, impartindu-I la cea mai mare valoare, 255.

X = X / 255

Astfel, setul de date a fost adus la valoarea optima.

4. Antrenarea unui model bazat pe algoritmi de invatare automata

Pentru aceasta aplicatie am utilizat algoritmul KNN (K-Nearest Neighbor). Algoritmul K-Nearest Neighbors (KNN) este un algoritm de invatare supervizata folosit pentru clasificare si regresie.

Pentru clasificare, KNN se bazeaza pe ideea ca obiectele similare sunt in general in apropiere unul de celalalt in spatiul caracteristicilor. Pentru a clasifica un punct de date necunoscut, algoritmul cauta cei mai apropiați "k" vecini din setul de antrenare pe baza unei distante.

Am antrenat algoritmul KNN pe 10 valori ale lui k, de la 1 la 10, calculand pentru fiecare valoare acuratetea, pe baza subsetului de validare. Subsetul de validare are rolul de a ne ajuta sa atribuim cea mai buna valoare hyperparametrului “k”, cel care ne ofera cea mai mare acuratete.

Am folosit urmatoarea biblioteca din Python: KNeighborsClassifier (sklearn.neighbors).

Cod pentru antrenarea algoritmului:

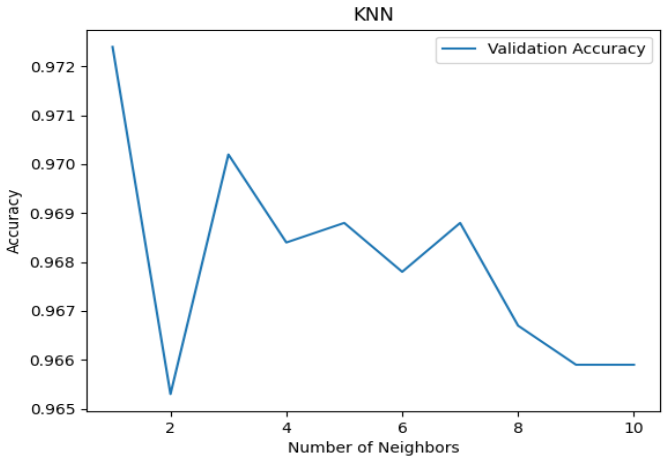
```
for k in range(1, 11):
    # train the k-Nearest Neighbor classifier
    model = KNeighborsClassifier(n_neighbors = k)
    model.fit(X_train, Y_train)
    # evaluate the model on validation dataset
    scoreValidation = model.score(X_validation, Y_validation)
    # validation results
    print("k = %d, accuracy = %.2f%%" % (k, scoreValidation * 100))
    accuracies.append(scoreValidation)
```

5. Afisarea si interpretarea rezultatului antrenarii

Pentru fiecare valoare a lui k intre 1 si 10 am obtinut urmatoarea acuratete:

```
k = 1, accuracy = 97.24%
k = 2, accuracy = 96.53%
k = 3, accuracy = 97.02%
k = 4, accuracy = 96.84%
k = 5, accuracy = 96.88%
k = 6, accuracy = 96.78%
k = 7, accuracy = 96.88%
k = 8, accuracy = 96.67%
k = 9, accuracy = 96.59%
k = 10, accuracy = 96.59%
```

Se poate observa cum pentru k = 1 avem cea mai buna acuratete, de 97.24%. Acest lucru a fost surprins si in urmatorul grafic:



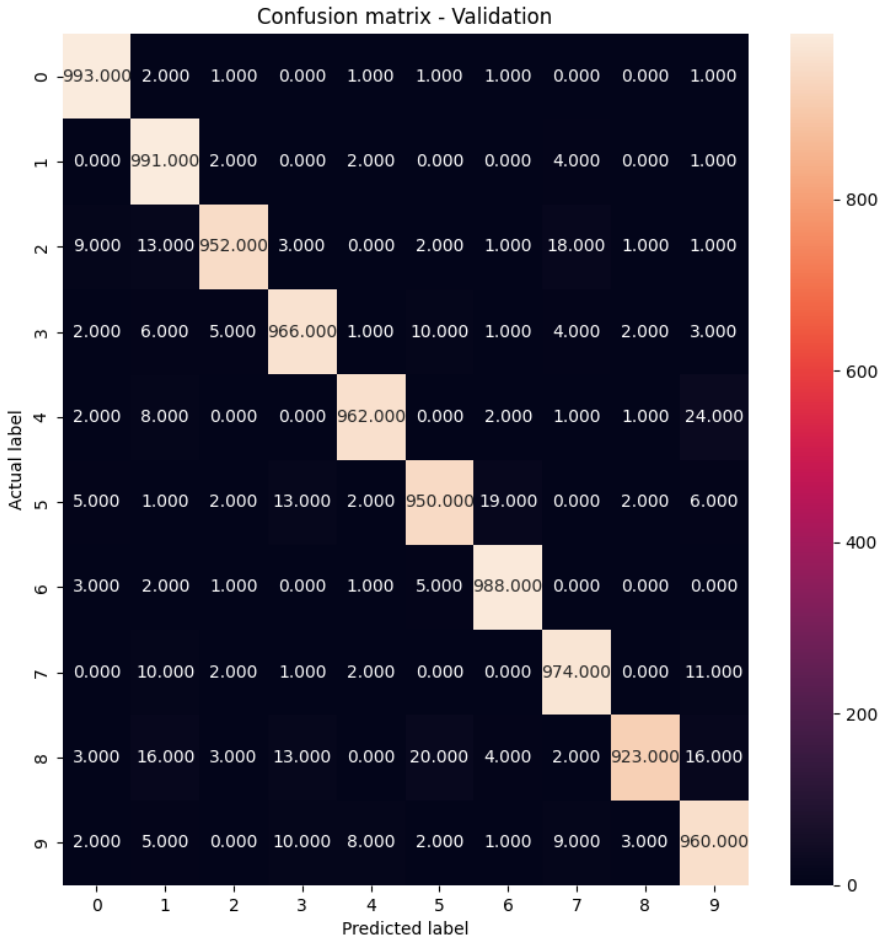
Astfel, cel mai buna valoarea a lui k pentru care se poate trece la etapa de testare este 1.

```
k = 1 : highest accuracy of 97.24% on validation dataset
```

Testand pentru cele 10 valori ale lui k, avem urmatorul timp de executie:

```
Execution time of training is:  67.706871 seconds
```

Avem urmatoarea matrice de confuzie, rezultata in urma antrenarii algoritmului, testand pe setul de validare:



Se poate observa faptul ca algoritmul a fost antrenat corespunzator, iar rezultatele pe setul de validare sunt foarte bune. Valorile de pe diagonala se apropie foarte mult de 1000 (maximul, acuratete 100%), iar valorile din restul matricei sunt foarte mici, cuprinse intre 0 si 24, predominant cu valori mici. Astfel, exista foarte putine confuzii. Avand totusi o dimensiune redusa setul de date de validare (1000 de imagini pentru fiecare folder (cifra)), se continua in etapa de testare.

6. Testarea modelului antrenat pe un set de date de test

Testarea algoritmului s-a facut pe un set de date nou, diferit de cele de antrenare si validare.

Cod pentru testare:

model = KNeighborsClassifier(n_neighbors = kValues[kMax])

model.fit(X_train, Y_train)

predictions = model.predict(X_test)

7. Afisarea si interpretarea rezultatelor testarii

Putem observa ca am obtinut un timp scurt de executie:

```
Execution time of testing is:  5.876043 seconds
```

Pentru fiecare cifra, avem urmatoarea acuratete:

```
Digit 0: Accuracy = 99.39%
Digit 1: Accuracy = 99.65%
Digit 2: Accuracy = 96.12%
Digit 3: Accuracy = 95.74%
Digit 4: Accuracy = 95.72%
Digit 5: Accuracy = 95.85%
Digit 6: Accuracy = 98.43%
Digit 7: Accuracy = 96.50%
Digit 8: Accuracy = 94.05%
Digit 9: Accuracy = 95.64%
```

Se poate observa cum pentru cifrele 0 si 1 avem cea mai mare acuratete, 99.39%, respectiv 99.65%, iar pentru cigra 8, cea mai mica, de 94.05%. Totusi, acuratetea este foarte buna pentru fiecare cifra, fiind foarte aproape de ideal, ceea ce inseamna ca etapele de antrenare, validare si testare au descurs bine, iar algoritmul a fost antrenat si testat corespunzator.

Astfel, avem urmatoarea acuratete globala:

```
Overall Accuracy: 96.75%
```

Acuratetea globala a algoritmului KNN pe setul de date MNIST este impresionanta, atingând 96.75%. Acest lucru inseamnă c modelul a prezis corect cifrele in aproape 97 din 100 de cazuri.

Urmatorul tabel de clasificare ne arata caracteristicile importante ale clasificarii, precum acuratetea, precizia, scorul F1, sensibilitatea:

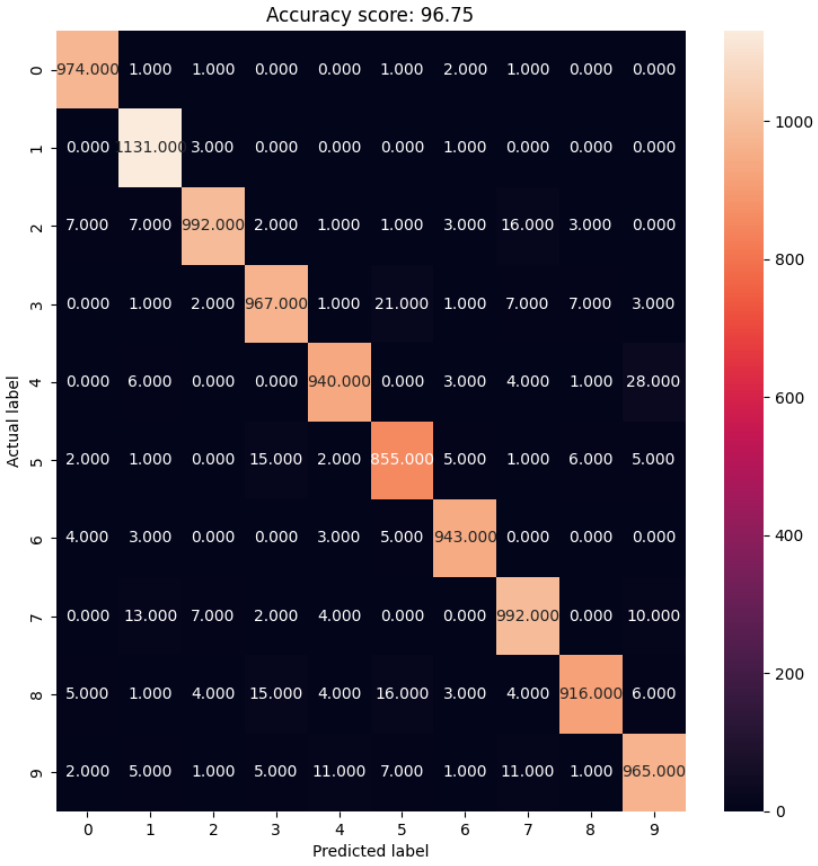
Classification Report

	precision	recall	f1-score	support
0	0.9798792756539235	0.9938775510204082	0.9868287740628167	980.0
1	0.9674935842600513	0.9964757709251101	0.9817708333333333	1135.0
2	0.9821782178217822	0.9612403100775194	0.9715964740450539	1032.0
3	0.9612326043737575	0.9574257425742574	0.9593253968253969	1010.0
4	0.9730848861283644	0.9572301425661914	0.9650924024640657	982.0
5	0.9437086092715232	0.9585201793721974	0.9510567296996664	892.0
6	0.9802494802494802	0.9843423799582464	0.9822916666666667	958.0
7	0.9575289575289575	0.9649805447470817	0.9612403100775194	1028.0
8	0.9807280513918629	0.9404517453798767	0.960167714884696	974.0
9	0.9488692232055064	0.956392467789891	0.9526159921026655	1009.0
accuracy	0.9675	0.9675	0.9675	0.9675
macro avg	0.9674952889885209	0.9670936834410779	0.967198629416188	10000.0
weighted avg	0.967625413582263	0.9675	0.9674654994290495	10000.0

- Modelul se comporta exceptional de bine pentru cifrele 0, 1, 2, 6 si 8, obtinand o precizie, o acuratete si un scor F1 de peste 95%. Cifrele 3, 4, 5, 7 si 9 au, de asemenea, o performanta buna, cu o precizie, acuratete si scor F1 de peste 94%.
- Chiar si cifra cu cea mai scazuta performanta (cifra 5) obtine totusi un scor F1 general de 95.11%.
- Metricile medie macro si medie ponderata ofera o masura generala a performantei. Ambele sunt apropiate de acuratetea globala, indicand o performanta echilibrata intre cifre.

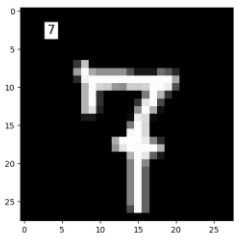
Matricea de confuzie este o metoda utila pentru a evalua performanta unui model de clasificare. Aceasta matrice ofera o imagine detaliata a rezultatelor predictiilor facute de model comparativ cu valorile reale din setul de date de testare.

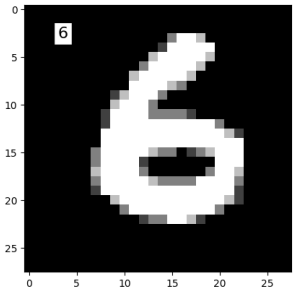
O matrice de confuzie ideala ar contine valori doar pe diagonala principala, restul fiind 0, asta ar inseamna ca nu au existat confuzii intre rezultatele din etapa testarii.

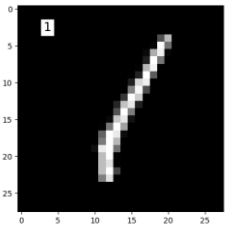


- Matricea de confuzie ofera insights cu privire la erorile specifice ale modelului. De exemplu, arata ca cifra 5 este adesea confundata cu cifrele 3, 8 si 9.
- Cifrele 3 si 8 prezinta o oarecare confuzie intre ele, precum si cu cifrele 5 si 9.

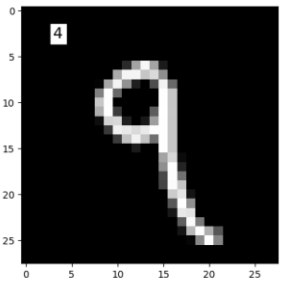
Pentru a testa corectitudinea algoritmului, am comparat valoarea prezisa cu imaginea originala din setul de testare. Astfel, avand imaginea, am putut observa ce cifra a prezis algoritmul ca este:





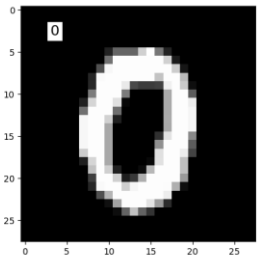


Putem observa ca exista si cazuri in care avem erori, deci cifra prezisa este diferita fata de cea din imagine:

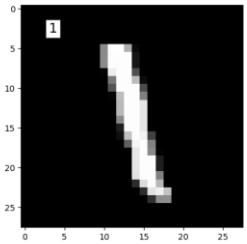


Pentru fiecare cifra din setul de date (0 .. 9), am evidentiat corectitudinea algoritmului. Anterior, am prezentat si cazuri in care cifra prezisa este diferita de cifra din imagine.

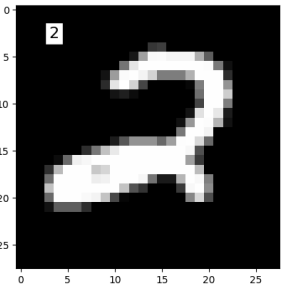
Cifra 0:



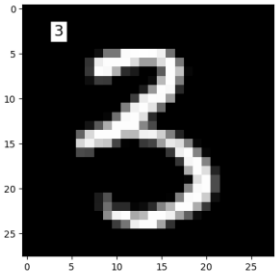
Cifra 1:



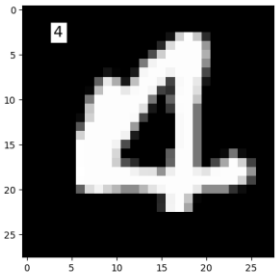
Cifra 2:



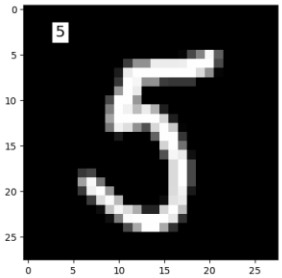
Cifra 3:



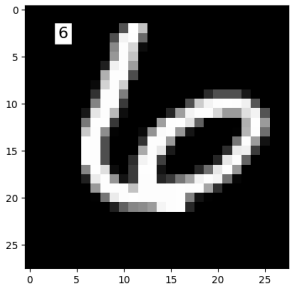
Cifra 4:



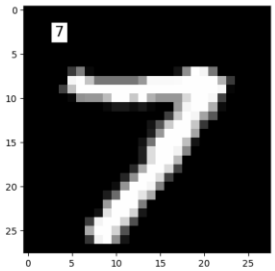
Cifra 5:



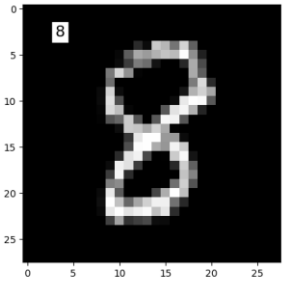
Cifra 6:



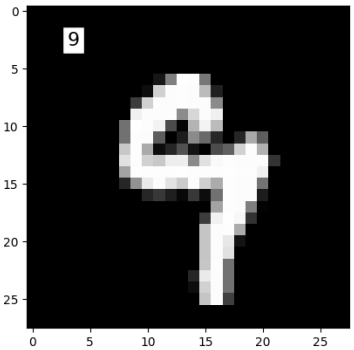
Cifra 7:



Cifra 8:



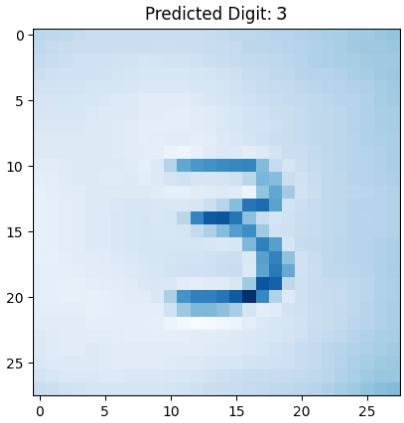
Cifra 9:



Pentru a testa corectitudinea algoritmului, am utilizat o imagine de test, scriind cifra 3 pe hartie. Fiind scris negru pe alb, am realizat o inversiune a culorilor, pentru a fi ca cele din setul de date, fundal negru, scris alb. Apoi, am redimensionat imaginea la dimensiunea de 28x28 pixeli.

Imaginea de testare:

Rezultatul in urma testarii:



Se poate observa ca programul a oferit rezultatul corect, cifra fiind 3.

8. Concluzii

- Algoritmul utilizat (KNN) ne ofera o acuratete de 96.75%, avand ca timp de testare de aproximativ 6 secunde, ceea ce face ca acest algoritm sa fie unul eficient pentru setul de date.
- Acuratetea inalta indica capacitatea algoritmului de a distinge cifrele scrise manual.

- Algoritmul se comporta foarte bine pentru majoritatea cifrelor (0-9), cu precizii si acurateti ridicate. Diferentele de performanta dintre cifre sunt minore, indicand o capacitate generala de clasificare.
- Exista confuzii notabile intre unele cifre, cum ar fi intre cifrele 3 si 8 sau intre cifrele 5 si 9.
- Prin evaluarea performantei pentru diferite valori ale parametrului k, s-a identificat k optim care maximizeaza acuratetea pe setul de validare. In acest caz, kMax a fost gasit pentru a fi un numar specific de vecini, asigurand o performanta optima.

Biblioteci utilizate:

```
import matplotlib.pyplot as plt

import numpy as np

from sklearn import datasets, metrics, svm

from sklearn.model_selection import train_test_split

from PIL import Image

from numpy import asarray

import os

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification_report

from sklearn.metrics import confusion_matrix

from datetime import datetime

import seaborn as sns

import pandas as pd
```